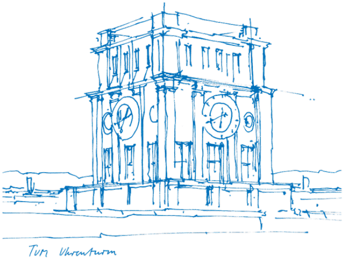
Honey for the Ice Bear – Dynamic eBPF in P4

Manuel Simon, Henning Stubbe, Sebastian Gallenmüller, Georg Carle

Sunday 4th August, 2024

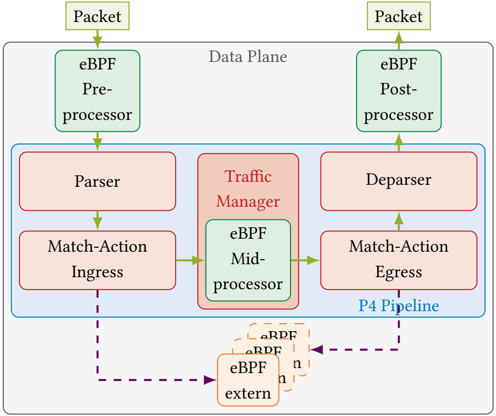
Chair of Network Architectures and Services
School of Computation, Information and Technology
Technical University of Munich



Motivation

- Interrupt-free, dynamic updates increase network resilience
  - ⇒ application migration
  - ⇒ tenant-specific processing
- P4 and eBPF are well-established languages for programmable packet processing
  - ⇒ P4: restricted, simple language, optimized for high performance
  - ⇒ eBPF: JIT compiled, more high-level language features
- Both languages bring advantages for specific use-cases
  - ⇒ eBPF programs as well-defined API for P4 externs to *extend* functionality

Dynmiac eBPF in P4



- Extension of P4 pipeline with updatable eBPF modules
  - Fixed position
  - Extern
- Allows runtime re-programmability
  - Exchange using pre-compiled **byte code**
  - JIT compiled to **machine code**
- Extends P4 functionality with well-defined API

Dynamic modes

Static

- Fixed, non-changeable functionality

Pre-defined

- Pre-implemented, fixed set of functionality
- Defined before *initialization*, switchable during *runtime*

Extensible

- New functionality is sent as source or byte code
- JIT compiled and bound during *runtime*

Related Work

Reprogrammable P4:

- Das et al., ActiveRMT [1]: Instruction set in P4 allowing changegable functionality
- Xing et al., FlexCore [6]: Runtime partial reprogrammable switch architecture
- Feng et al., In-situ Programmability Data Plane [3]: Switch architecture and reconfigurable P4 (rP4) for runtime updates

  ⇒ single-language P4 approaches

P4/eBPF:

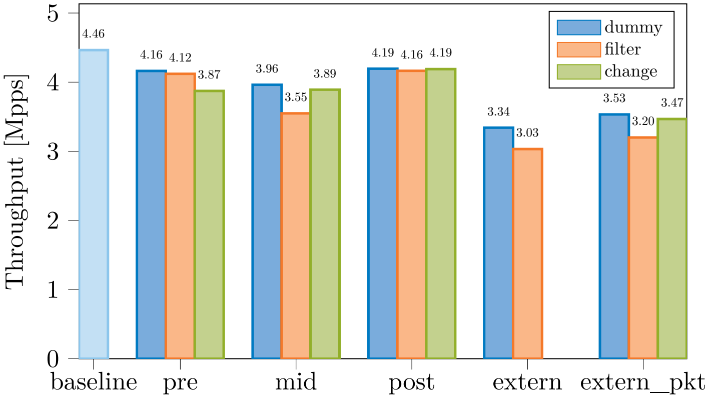
- P4 to eBPF [4]: Translation of P4 program to eBPF [4]

Implementation

- Implementation for software target *T4P4S* [5]
- eBPF execution using DPDK `rte_bpf` library
  - batched tx/rx eBPF callback execution for *fixed* position
  - non-batched execution for *flexible* externs
- User space eBPF execution
- Optional BLAKE3-based MACs ensuring authenticity of code updates

Evaluation

Overhead of eBPF execution at different positions (Throughput)

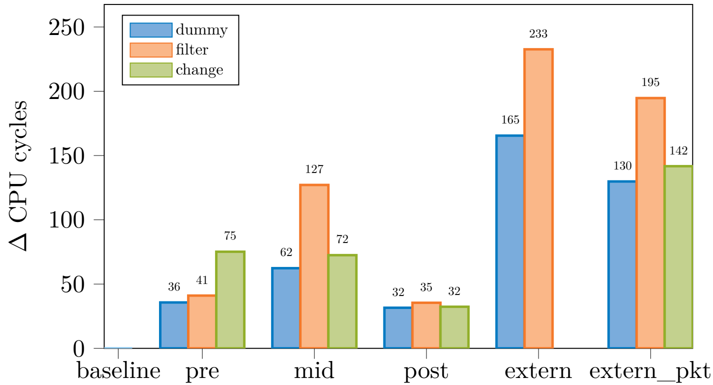


Three programs for basic overhead:

- *dummy*: returns 0
- *filter*: filters for one UDP port and IP address
- *change*: changes a header field

Evaluation

Overhead of eBPF execution at different positions (modeled per-packet CPU cycles)



Three programs for basic overhead:

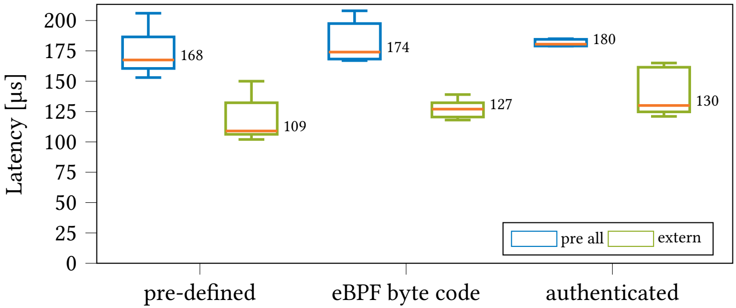
- *dummy*: returns 0
- *filter*: filters for one UDP port and IP address
- *change*: changes a header field

Cost model:

$$C = \frac{f_{CPU}}{r_{testbase}} - \frac{f_{cpu}}{r_{baseline}}$$

Evaluation

Median costs of dynamic updates—ten runs (100 Mbit/s)



⇒ Update of fixed-position functionality more expensive

⇒ Dynamic eBPF byte code installation at reasonable costs

⇒ Authentication possible

Conclusion

- eBPF offers fixed API for P4 externs
- eBPF hardware offloading solutions exist
- eBPF execution within P4 allows additional applications
- Functionality can be updated during runtime (200 µs)

Read the paper if you want more information about:

- Security considerations
- Discussion of different processor positions
- Detailed analysis of program change

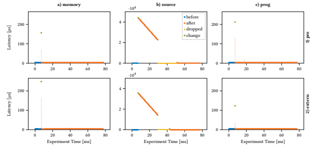


Figure 4: Latencies before, during, and after change packet

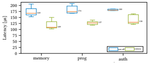


Figure 5: Latency of change packet (median as number)

eBPF is already performed on SmartNICs [3]. Implementing externs inside the P4 pipeline becomes more challenging due to the fixed clock rates between the pipeline stages. Conditional eBPF execution would likely not result in a performance gain for the packets that do not use the extern. Instead, the latencies are likely to be constant, independent of the executed control flow of each packet [18]. Another difficulty is to synchronize the execution times required for each stage. For that, the existing validators for eBPF guaranteeing maximum cycle counts can help. The same requirements hold for extensible updates, and maximum cycles have to be defined. Again, validators can help calculate maximum cycles. This requirement is more relaxed in software targets, i.e., run-to-completion targets.

eBPF helps ease runtime adaptability for hardware and software targets. P4 is a domain-specific language designed for packet processing exclusively; therefore, its execution can be optimized in hardware targets. The results show that providing new functionality by distributing its source code is not feasible. Conversely, a natively compiled binary can only be used on one specific platform. As eBPF can be distributed as platform-independent byte code and, as the results show, be installed in a timely manner, it is well-suited to enable such mechanisms. The same binary can be used for all targets, hardware, or software. The JIT compilation unleashes full performance, optimizing it for the underlying machine architecture.

7 Conclusion

We discussed, implemented, and evaluated different approaches to offload eBPF execution within P4. The overhead is smaller for fixed-position components than for flexible externs. Fixed-position components are likely easier to integrate into hardware targets. However, externs are more flexible in their usage. For dynamic changes, the fastest option is to activate pre-defined eBPF programs. However, the more powerful extensible updates, relying on eBPF binaries, are feasible. Dynamic updates allow an interrupt free service of the network. A dynamic network function can be implemented and secured, leveraging authenticated updates. On the other hand, sending dynamic updates using the source code proved impractical due to the significant compilation overhead, which eventually causes packet loss.

The results demonstrate that eBPF execution with dynamic and seamless updates is possible, enabling a variety of new applications. The source code of our implementaion is available on GitHub [26].

Acknowledgments

The authors thank Timon Tsiolis for his contributions and the fruitful discussions, and the reviewers for their valuable feedback. This work is partially funded by the European Union's Horizon 2020 research and innovation programme (grant agreement no. SLICES-PP 101079774 and GreenDIGIT 101131207). The German Federal Ministry of Education and Research (BMBF) supported our work under the projects 6G-life (16KISK002) and 6G-ANNA (16KISK107) as well as the German Research Foundation (DFG) as part of the HyperNIC (CA595/13-1) project.

Bibliography

[1] R. Das and A. C. Snoeren.
Memory management in activermt: Towards runtime-programmable switches.
In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 10431059, New York, NY, USA, 2023. Association for Computing Machinery.

[2] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle.
Moongen: A scriptable high-speed packet generator.
In K. Cho, K. Fukuda, V. S. Pai, and N. Spring, editors, *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, pages 275–287. ACM, 2015.

[3] Y. Feng, Z. Chen, H. Song, W. Xu, J. Li, Z. Zhang, T. Yun, Y. Wan, and B. Liu.
Enabling in-situ programmability in network data plane: From architecture to language.
In A. Phanishayee and V. Sekar, editors, *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, pages 635–649. USENIX Association, 2022.

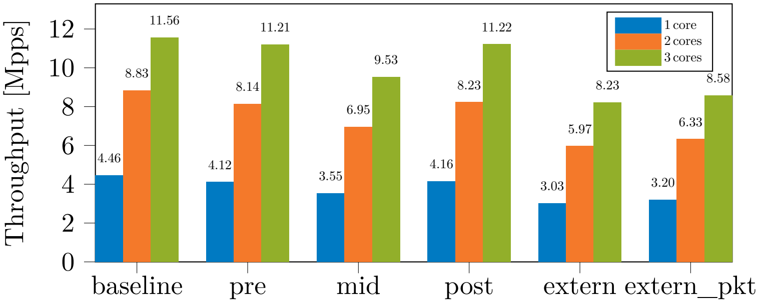
[4] p4lang.
GitHub p4c/backends/ebpf - eBPF Backend, 2024.
Last accessed: 2024-05-24.

[5] P. Vörös, D. Horpácsi, R. Kitlei, D. Leskó, M. Tejfel, and S. Laki.
T4P4S: a target-independent compiler for protocol-independent packet processors.
In *IEEE 19th International Conference on High Performance Switching and Routing, HPSR 2018, Bucharest, Romania, June 18-20, 2018*, pages 1–8. IEEE, 2018.

[6] J. Xing, K.-F. Hsu, M. Kadosh, A. Lo, Y. Piasetzky, A. Krishnamurthy, and A. Chen.
Runtime programmable switches.
In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 651–665, Renton, WA, Apr. 2022. USENIX Association.

BACKUP

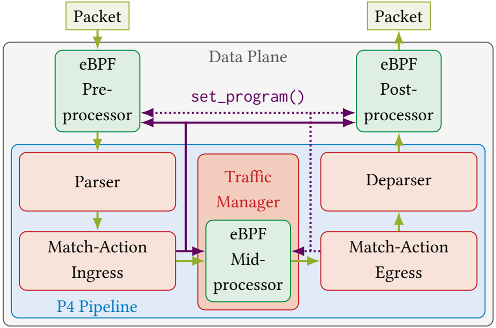
BACKUP



Throughput [Mpps]
0
2
4
6
8
10
12
1 core
2 cores
3 cores
4.46
8.83
11.56
4.12
8.14
11.21
3.55
6.95
9.53
4.16
8.23
11.22
3.03
5.97
8.23
3.20
6.33
8.58
baseline
pre
mid
post
extern
extern_pkt

BACKUP

Dynamic eBPF in P4

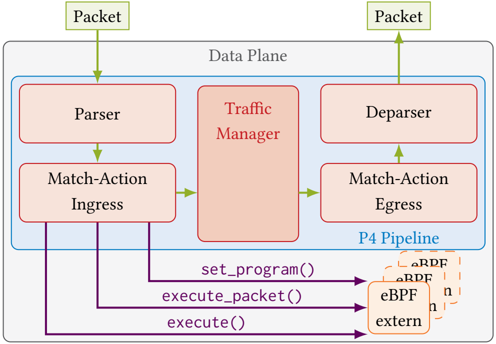


Fixed position

- *Pre-*, *Mid-*, or *Postprocessor*
- Processes every packet
- Access to whole packet
- Potentially easier implementation
- E.g., prefilter, preprocessing, hashing/crypto

BACKUP

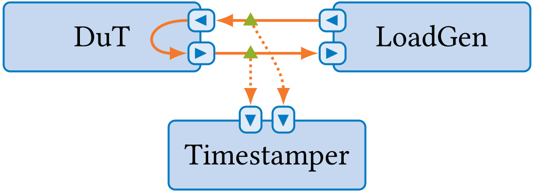
Dynamic eBPF in P4



Extern

- Flexible position as P4 extern
- Conditional execution
- Return value usable
- Access to whole packet *or* restricted to selected header fields

Setup



DuT

- Intel Xeon D-1518 2.2 GHz, 32 RAM
- Latency optimized *T4P4S* → batch size of one

LoadGen

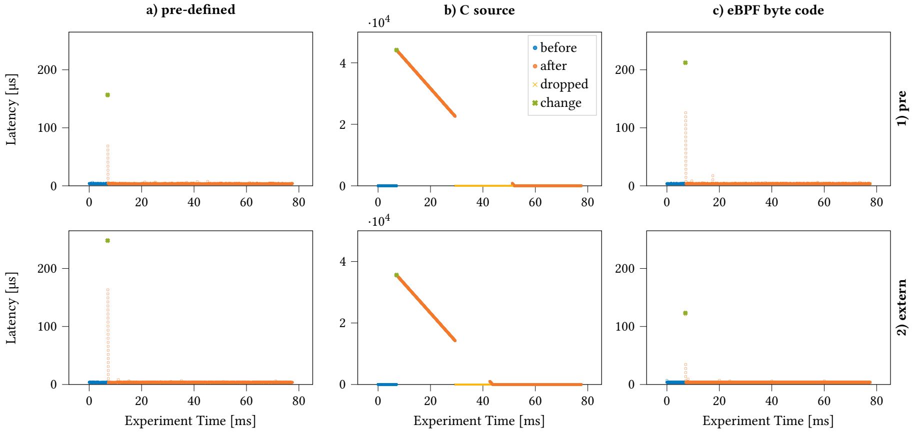
- MoonGen [2] is used to generate traffic
- Packet size 84 B

Timestamper

- Packet streams duplicated using optical splitter
- Timestamps each packet incoming packet
- Resolution: 12.5 ns

BACKUP

Costs of dynamic updates—single run (100 Mbit/s)



⇒ eBPF byte code swapping during runtime possible without packet loss